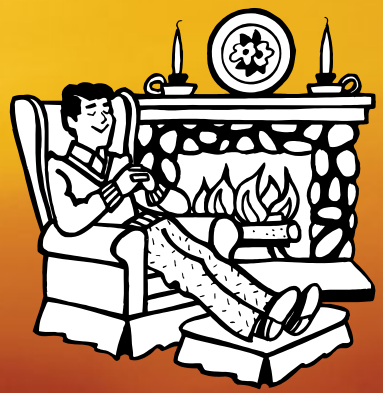
PDC03
Make the connection

A Fireside Chat: Building Software for a Hostile Environment

Michael Howard
mikehow@microsoft.com
Senior Program Manager
Security Engineering & Communications
Microsoft Corporation



Trends

Enterprise Agility

- Creating a mobile enterprise by providing employees
with quick and secure access to information and services they need

“60 Million mobile workers in the US today”

“63.4M handheld computers to be sold by 2003”

“US Mobile workforce to nearly double between 2001-2006 ”

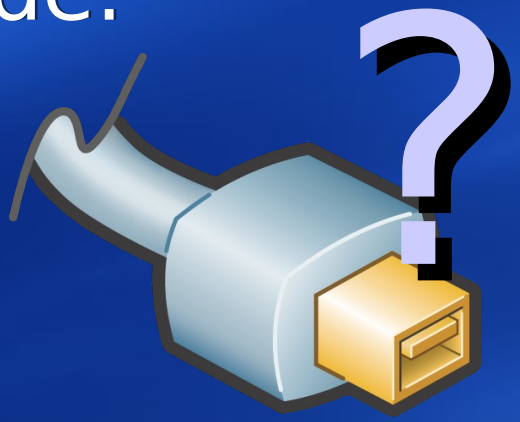
Source: Gartner and IDC

Scenarios

- Employees connecting to company's network
  - Wired, Wireless, Dial-up, VPN
  - Company's PCs, Personally owned systems
- Employees connecting to other networks
  - Internet hotspots, partner networks, broadband
- Partners connecting to company's net
  - Local vs. federated authentication
  - Anonymous guests
- New Scenarios & New Threats

Folks, The Threats have Changed!

- The Internet is the seediest neighborhood on the planet
- Your neighbors include:
  - Thieves
  - Con-artists
  - Vandals
  - Criminals
  - Hackers
- Do you wonder why attacks occur?



PDC03
Make the connection

Attack!

demo

Michael Howard

Software Must Adapt

- We must adapt as new threats arise
- Competitive ‘threats’
  - We all embraced 16-bit Windows
  - We all migrated to 32-bit Windows
  - We welcomed the Internet
  - We warmed to XML Web Services
- We must adapt to a less trusting, more hostile environment
- Software from the ‘nice end of town’ won’t survive long in the slums of the Internet

The Role of Security

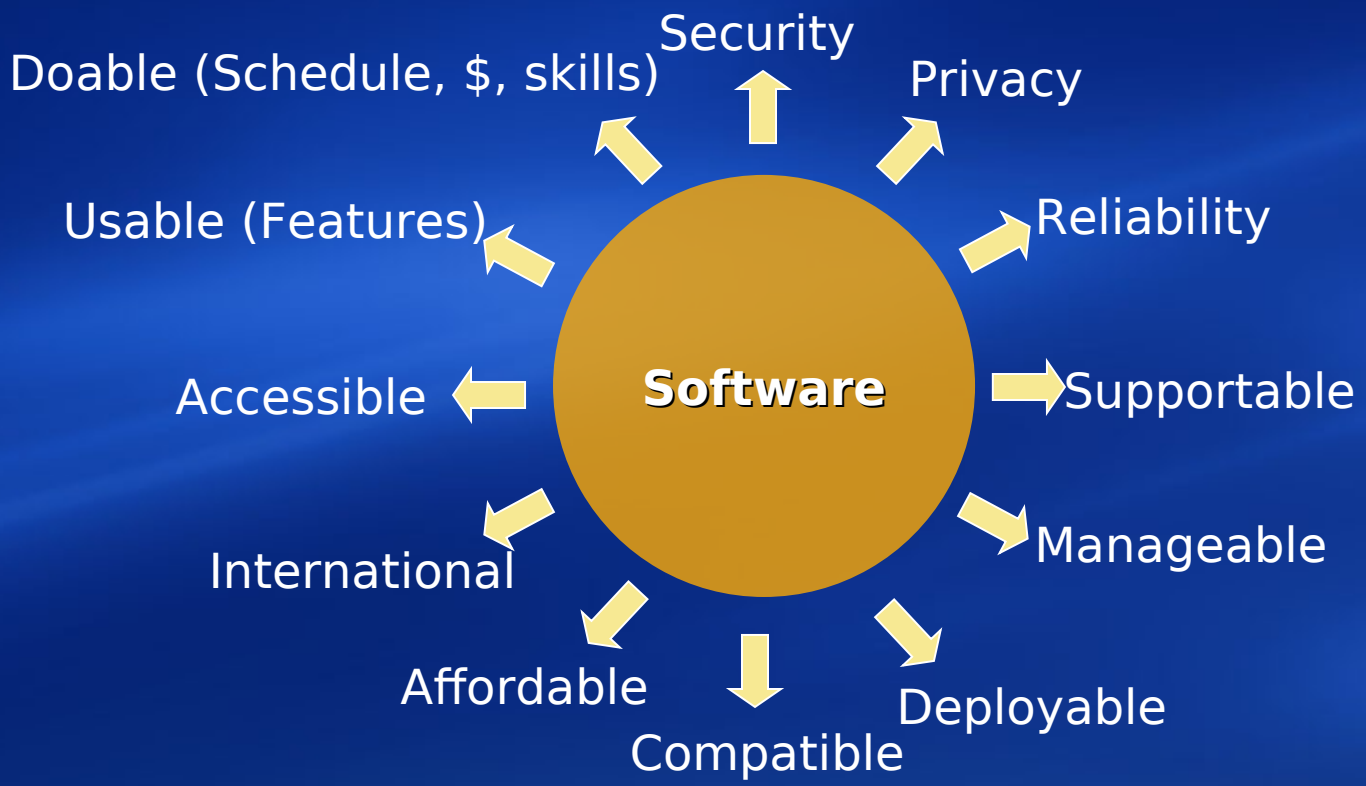
- Security is the current ‘big thing’
- Security skills are in very short supply
- Ignorance is not bliss!
  - Not for you, anyway!
- This is your opportunity to adapt faster than your competition

The Attacker's Advantage and the Defender's Dilemma

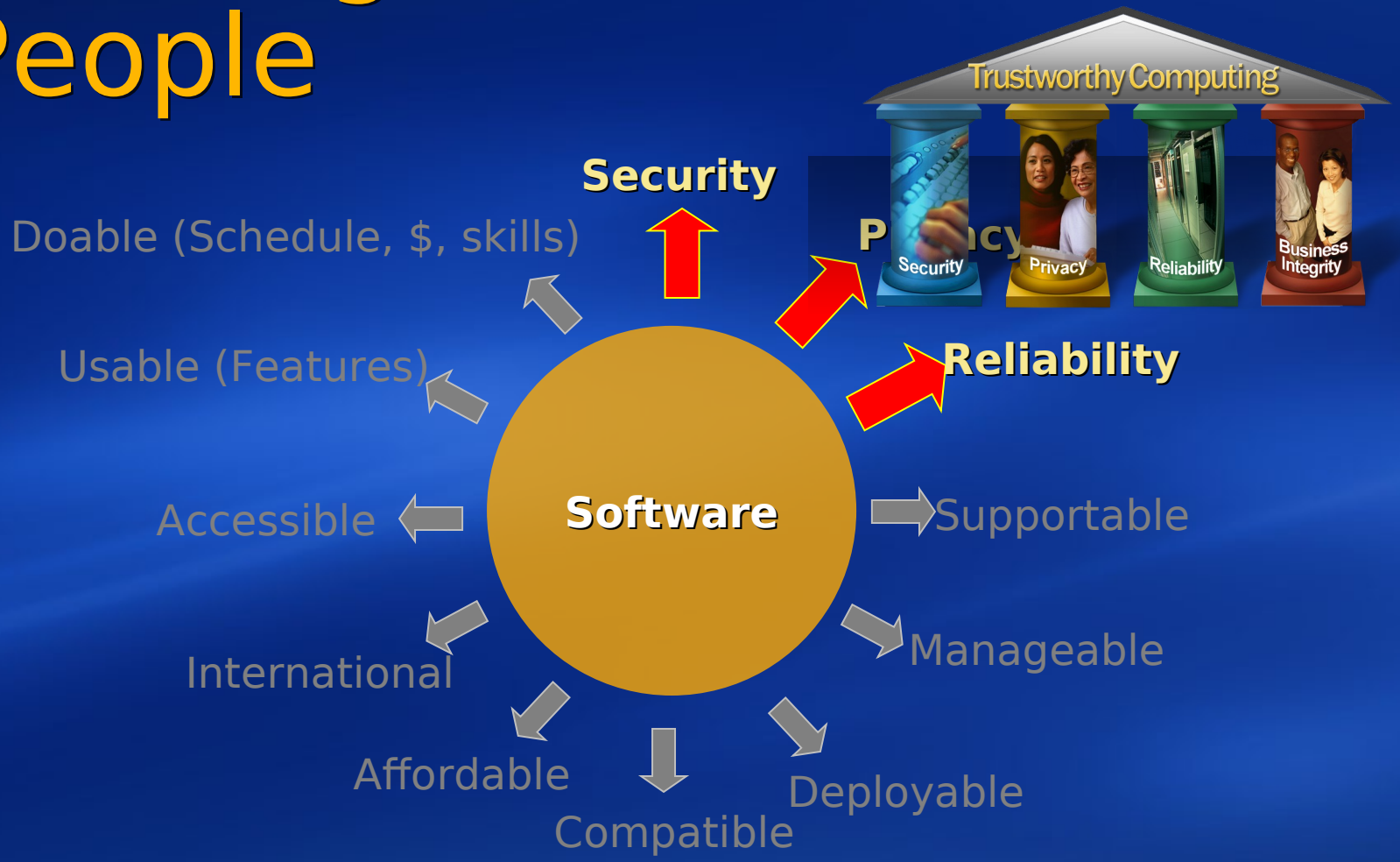
- Principle #1 The Defender must defend all points; the Attacker will choose the weakest point
- Principle #2 The Defender must be constantly vigilant; the Attacker will strike at will
- Principle #3 The Defender can only defend what he/she knows about; the Attacker will study for vulnerable points

Build it – and the attackers will come!

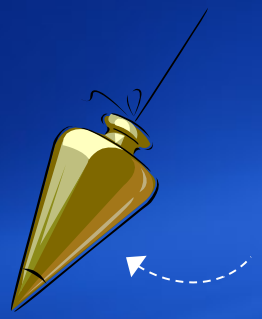
Building Software for People



Building Software for People



The Software Security Pendulum - Where we were



Usability & Features

Security & Privacy

- Easy to use
- Large attack surface
- ‘Automagic’
- Internet Time
- Things (and attacks) ‘just work’

The Software Security Pendulum - Where we are

- Reduced attack surface
- Often harder to use
- “Knee Jerk” security
- Slower to market
- Many security prompts

The Software Security Pendulum - Where we should be

Usability & Features

Security & Privacy

- Small attack surface
- Security & privacy a first-class feature
- Few security prompts
  - Transparent where possible
  - Configurable where necessary
- Securely usable!

Developers: *A Call to Action*

Secure by Design

- A Process that fosters secure systems
- Build threat models
- Conduct code reviews, penetration tests
- Run code with minimal privileges

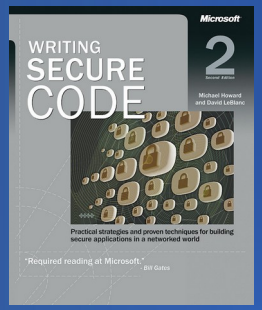
Secure by Default

- Minimize your attack surface

  (services off by default)
- Enable services securely
- Use security features in Visual Studio .NET

Secure in Deployment

- Leverage the security best practices from the platform
- Create security guidance
- Build tools to assess application security



Writing Secure Code 2

Microsoft Developer Network

http://msdn.microsoft.com/security

Patterns and Practices Guides

http://www.microsoft.com/patterns

The Rest of the Security Symposium

- Implementing the Call to Action
  - Practice
  - Process
  - Solutions

PDC03

Make the connection

Security Practice and You

Michael Howard
mikehow@microsoft.com
Senior Program Manager
Security Engineering & Communications
Microsoft Corporation

Microsoft

What's Wrong with this Code?

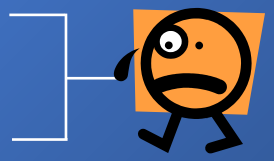
Port 135 (i.e. The Internet)

```
error_status_t _RemoteActivation(WCHAR *pwszObjectName,    ... )  {
    *phr = GetServerPath( pwszObjectName, &pwszObjectName);
    ...
}

HRESULT GetServerPath(WCHAR *pwszPath, WCHAR **pwszServerPath ){
    WCHAR * pwszFinalPath = pwszPath;
    WCHAR wszMachineName[MAX_COMPUTERNAME_LENGTH_FQDN + 1];
    hr = GetMachineName(pwszPath, wszMachineName);
    *pwszServerPath = pwszFinalPath;

}

HRESULT GetMachineName(
    WCHAR * pwszPath,
    WCHAR   wszMachineName[MAX_COMPUTERNAME_LENGTH_FQDN + 1]) {
    pwszServerName = wszMachineName;
    LPWSTR pwszTemp = pwszPath + 2;
    while ( *pwszTemp != L'\\' )
        *pwszServerName++ = *pwszTemp++;
```



The Core Error

- Copying untrusted data
- **while (*c!=somechar) *p++ = *c++** is constrained by source data not by destination size

Constrained only by source data

```
while (*c != '\\')
    *p++ = *c++;
```

Came from the network!

There are Two Major Kinds of Security Defects

1. Input trust issues
2. Everything else

Input trust issues

- Buffer Overruns

- SQL Injection

Blake	Blake’ drop table foo --

- Cross-Site Scripting

Blake	<script>var i=document.cookie</script>

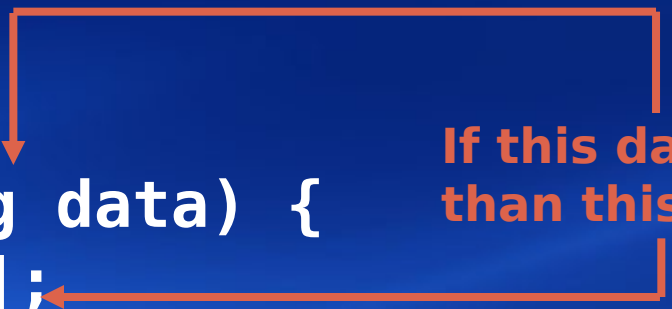
What are Buffer Overruns?

- External data is larger than the destination
- Overflowing the destination tramples some sensitive in-memory construct that determines execution flow
  - Causing the application to change execution flow
  - To the attacker's code included in the data
- C/C++ code the most common victim
  - Direct access to memory
- Cause: trusting input

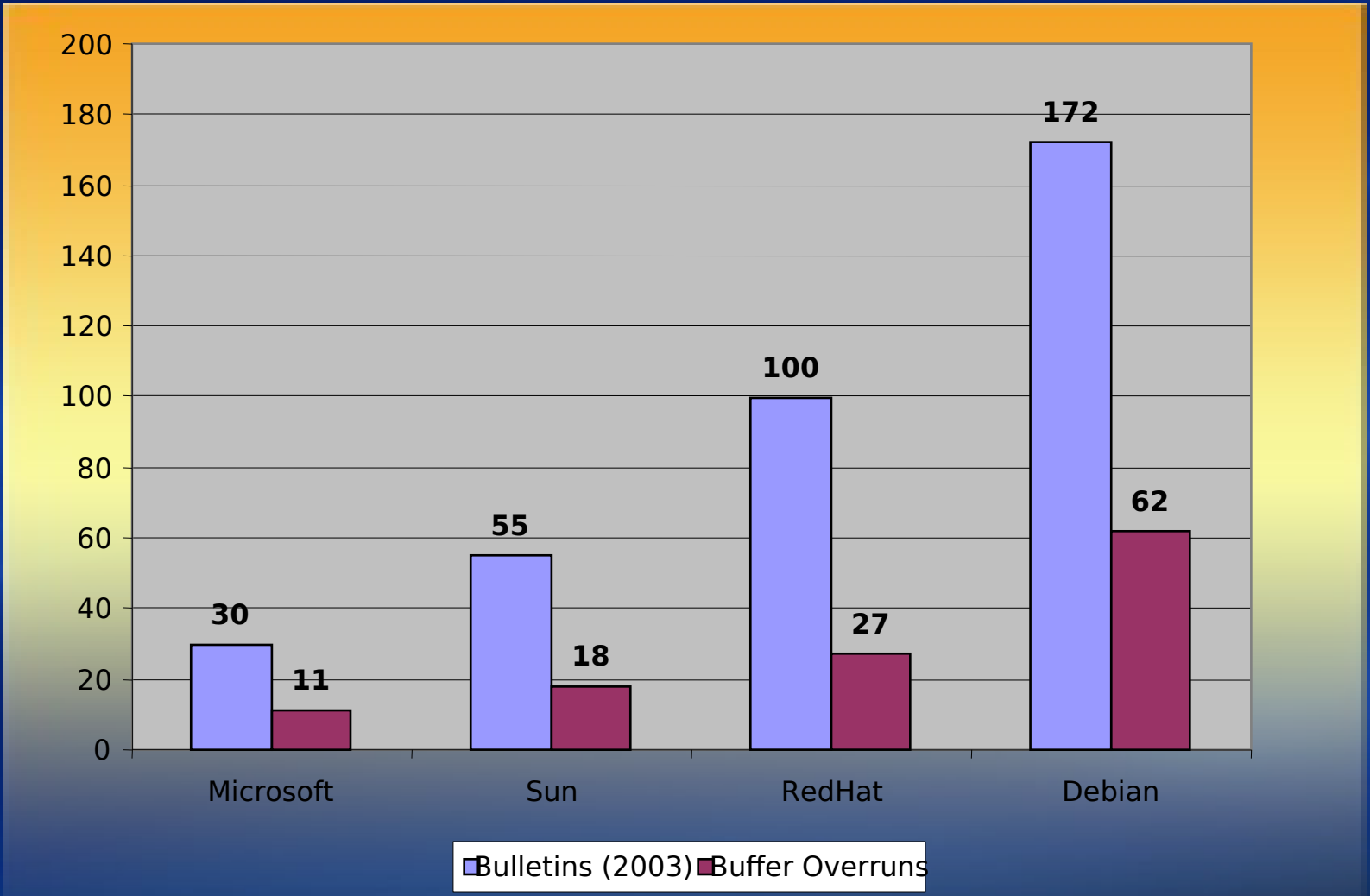
Example

```
void CopyStuff(string data) {
     char buffer[128];
     strcpy(buffer,data);

     // do other stuff
}
```



How common are BOs?



Source: Vendor Web Sites
Timeframe: 01Jan2003 – 01Oct200

Why are they prevalent?

- Lots of C/C++ code out there
- Lots of legacy C/C++ code that's now hooked to the internet
- Many data structures jump to code
  - Stack addresses, function pointers, exception handlers, C++ class v-tables etc.
- Constantly evolving threats
  - First there were stack overruns
  - Then heap overruns
  - Then format string overruns
  - Then "one-byte overruns"
  - Now, integer overflows
  - What's next?
- Developers don't check the input data!

Integer Arithmetic Vulns

```
int ConcatString(char *buf1, char *buf2,
                 size_t len1, size_t len2){
    char buf[256];
    if((len1 + len2) > 256) return -1;
    memcpy(buf, buf1, len1);
    memcpy(buf + len1, buf2, len2);
```

What if: len1 == 0xFFFFFFFE
len2 == 0x00000102

0xFFFFFFFE + 0x00000102 == 0x100

Integer Arithmetic Vulns

- Once rare, now common
  - Sun RPC xdr_array (http://www.securityfocus.com/bid/5356)
  - OpenSSH authentication (http://www.securityfocus.com/bid/5093)
  - Apache Chunked Encoding (http://www.securityfocus.com/bid/5033)
  - Microsoft JScript (http://www.microsoft.com/technet/security/bulletin/MS03-008.asp)
  - FreeBSD socket and system calls (http://www.securityfocus.com/bid/5493)
  - Snort TCP Packet Reassembly (http://www.securityfocus.com/bid/7178)
  - Microsoft MIDI Decoder (http://www.microsoft.com/technet/security/bulletin/MS03-030.asp)
  - OpenSSL ASN.1 Parsing

Remedies: Training

- Ongoing
  - Increased focus on code that:
    - Accesses the network
    - Runs by default
    - Uses unauthenticated protocols
    - Runs with elevated privileges
- Writing Secure Code 2nd Ed
- http://msdn.microsoft.com/security
- http://www.microsoft.com/patterns

Remedies: Enforcing Less Error-prone Constructs

- C runtime has many ‘dangerous’ function calls
  - gets, strcpy, strcat, sprintf(“%s”) etc
- Simply replacing a ‘dangerous’ function with a ‘safe’ function does not make the code safe
  - You may get the buffer size wrong
  - Can introduce regressions
- strsafe created during Windows Security Push
  - Actively employed in Windows Server 2003 and later
- Safer CRT
- Stop using C runtime buffer/string functions in C++ code!
  - Use C++ STL string class
- Be wary of arithmetic that calculates buffer sizes

PDC03
Make the connection

Safer CRT & strsafe.h

demo

Michael Howard

Remedies: Tool Usage

- The best tool is between your ears!
- Build tools to scale the problem
  - Alone, tools do not secure software make
- Compiler and OS mitigation
  - /GS compiler option and Safe Exception handling
  - Non-executable pages

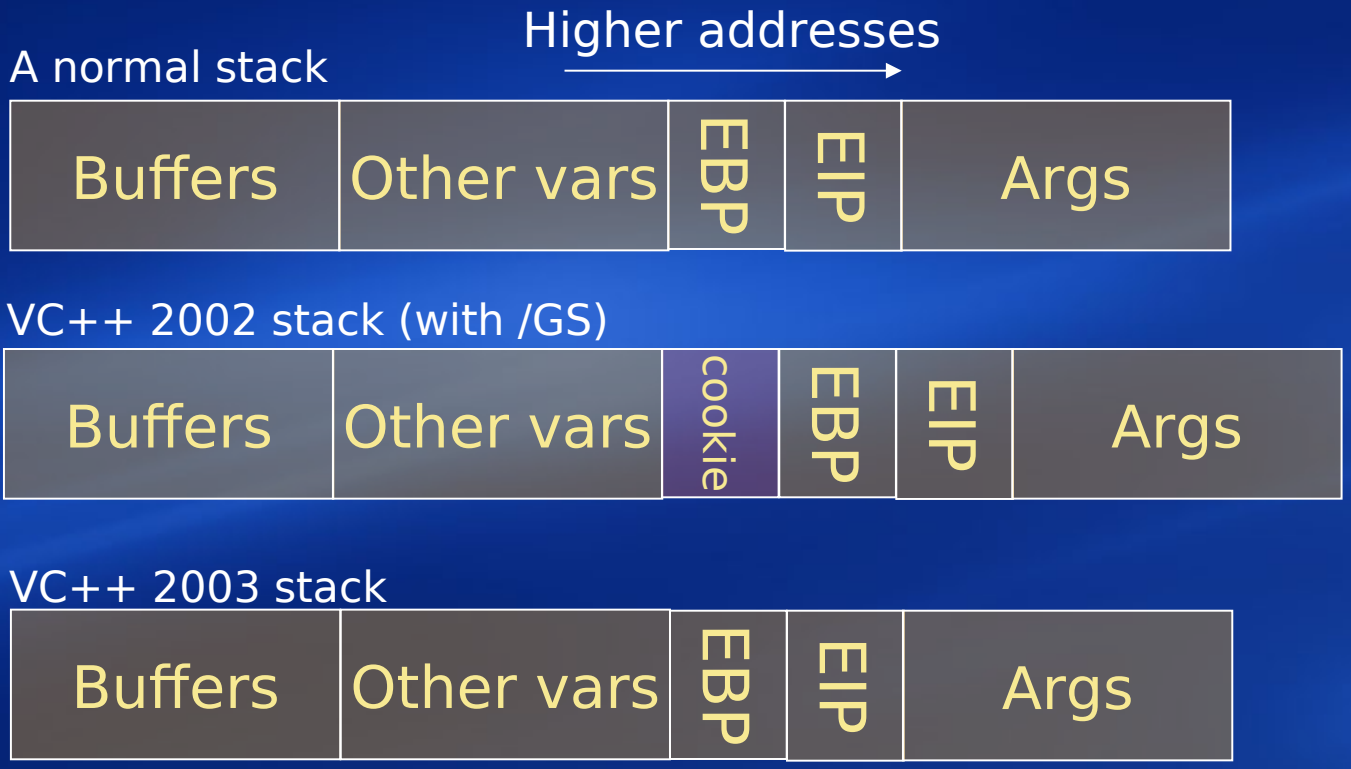
VC++ 2002 /GS Defense at work

A normal stack

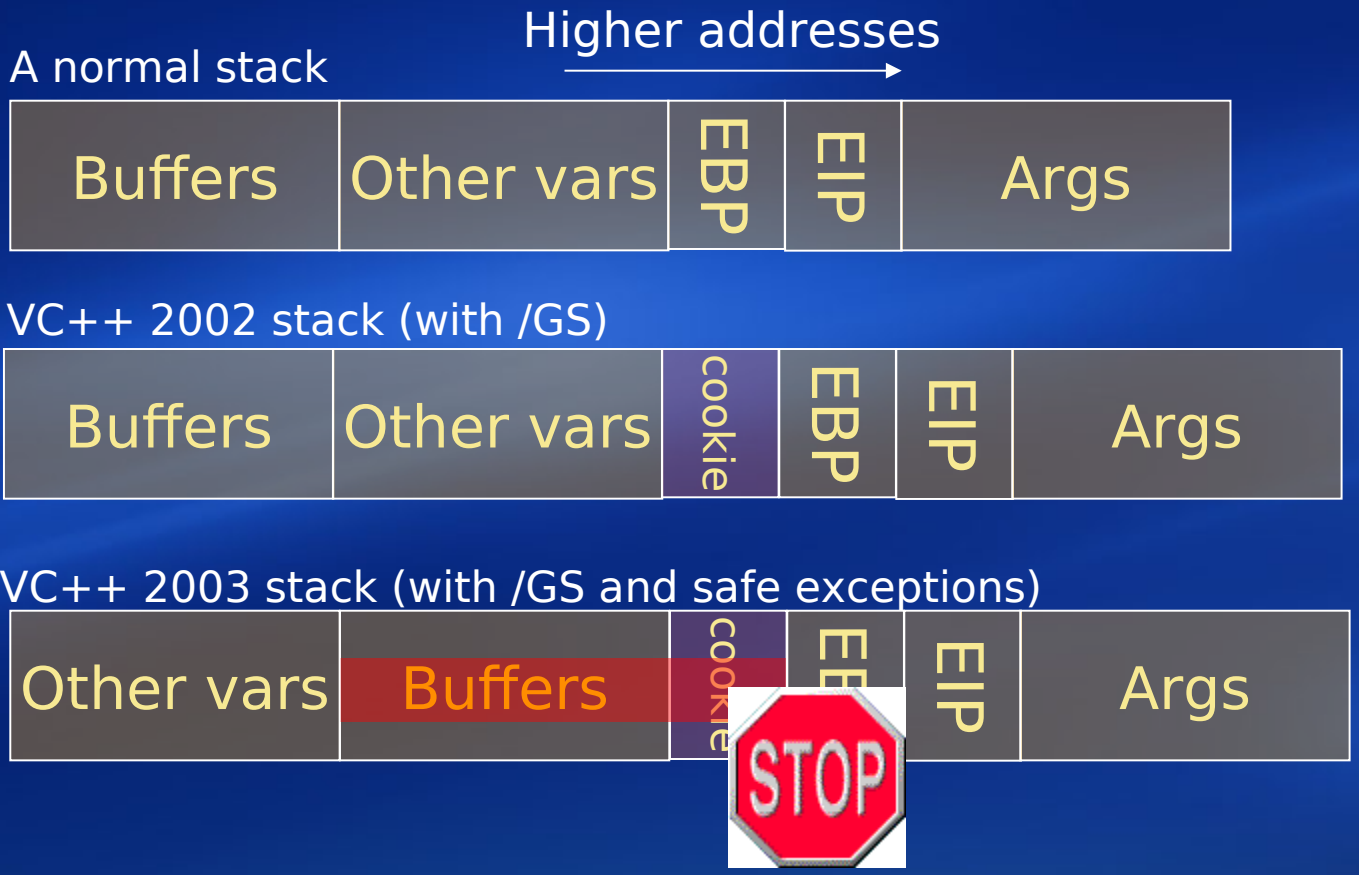
Higher addresses

VC++ 2002 stack (with /GS)

VC++ 2003 /GS Defense at work



VC++ 2003 /GS Defense at work



Remedies: Higher-level languages

- Consider migrating C/C++ code higher-level languages ***as appropriate***
- UDDI Service in Windows Server 2003 is written in C#

A Best Defense

- Reduce your attack surface
  - Turn the feature off
  - Run with lower privilege
- The Attack Surface Reduction Process
  1. Review the code
  2. Fix the code
  3. Compile with /GS
  4. Run as non-admin
  5. THEN TURN THE FEATURE OFF!

Summary

- There is no “silver bullet” to create secure software
- Requires remedies and defenses
- Requires process improvements

PDC03
Make the connection

Matt Lyons

mattlyon@microsoft.com
Program Manager
Security Engineering & Communications
Microsoft Corporation

Managed Code and Buffer Overruns

- Now it's hard to *accidentally* overwrite arbitrary memory

```
private void CopyStuff(string data) {
      char[] buffer = new char[128];
      data.CopyTo(0, buffer, 0, data.Length);
      // do other stuff
}
```

If this data is larger than this buffer

Then this function call will cause an exception

Managed Code and Integer Arithmetic

- Buffer overrun protections help reduce severity of integer overflows

```
byte[] allData;  // class member variable


public void BeginCopy(byte[] KnownBytes, int BytesToGo) {
  allData = new byte[KnownBytes.Length + BytesToGo];
  Array.Copy(KnownBytes, allData, KnownBytes.Length);
}
```

At worst, this function call will cause an exception instead of overwriting arbitrary memory

Integer overflow here will cause exception

So All Input Can Be Trusted In Managed Code, Right?

- Unfortunately, no
  - “Managed” does not mean absolute security
  - “Code Access Security” does not mean all managed code is secure
- “All input is evil until proven otherwise”

Managed Code and Cross-Site Scripting

- ASP.NET provides some built-in mitigations against cross-site scripting attacks
- The mitigations are not absolute – input validation should still be done
- Solution: validate all untrusted input before processing

PDC03
Make the connection

ASP.NET Cross-Site Scripting Protections

demo

Matt Lyons

Integer Arithmetic

- Integer arithmetic can still overflow and underflow in managed code
- Solutions: verify arithmetic, apply ranges to input, use “checked” semantics in C#

```
// 0-19 system data, 20-499 user data
Object[] myData = new Object[500];

private Object GetSystemData(uint ItemNumber) {
  return myData[ItemNumber];
}
public Object GetUserData (uint ItemNumber) {
  return myData[ItemNumber + 20];
}
```

What if ItemNumber Is UInt32.MaxValue?

Remoting

- Remoting currently has two built-in channels: HTTP and TCP
  - HTTP channel works through IIS
  - TCP channel works through sockets
- Remoting has no built-in authentication or encryption support itself
- Solution: Use HTTP channel to leverage IIS or use channel sinks
  - http://msdn.microsoft.com/library/en-us/dnnetsec/html/SecNetCh11.asp

SQL Injection

- There are no default mitigations for SQL injection attacks
- Solution: validate input and use parameterized commands

PDC03
Make the connection

SQL Injection

demo

Matt Lyons

The “Turkish-İ problem” (Applies also to Azerbaijan!)

- Turkish has four letter ‘I’s
  - i (U+0069) I (U+0049) ı (U+0131) İ (U+0130)
- In Turkish locale UC("file")==FİLE

```
// Do not allow "FILE://" URLs
if(url.ToUpper().Left(4) == "FILE")
  return ERROR;
getStuff(url);
```

```
// Only allow "HTTP://" URLs
if(url.ToUpper(CultureInfo.InvariantCulture).Left(4) == "HTTP")
  getStuff(url);
else
  return ERROR;
```

Call To Action

- Use managed code when appropriate
  - It prevents or mitigates some of today's common secure coding problems
- Continue to validate untrusted data
  - Test against what you know is good, not what you think is bad

Resources

- .NET Framework Security
- Writing Secure Code 2nd Ed
- http://msdn.microsoft.com/columns/secure.asp
  - “An Overlooked Construct and an Integer Overflow Redux”
  - “Reviewing Code for Integer Manipulation Vulnerabilities”
  - “Fending Off Future Attacks by Reducing Attack Surface”
  - “Fix Those Buffer Overruns!”
- http://msdn.microsoft.com/net/security

PDC03

Make the connection

Security Process and You
SQL Server Case Study

James Hamilton
jamesrh@microsoft.com
General Manager - SQL Server Web data
Development & Security Architect
Microsoft Corporation

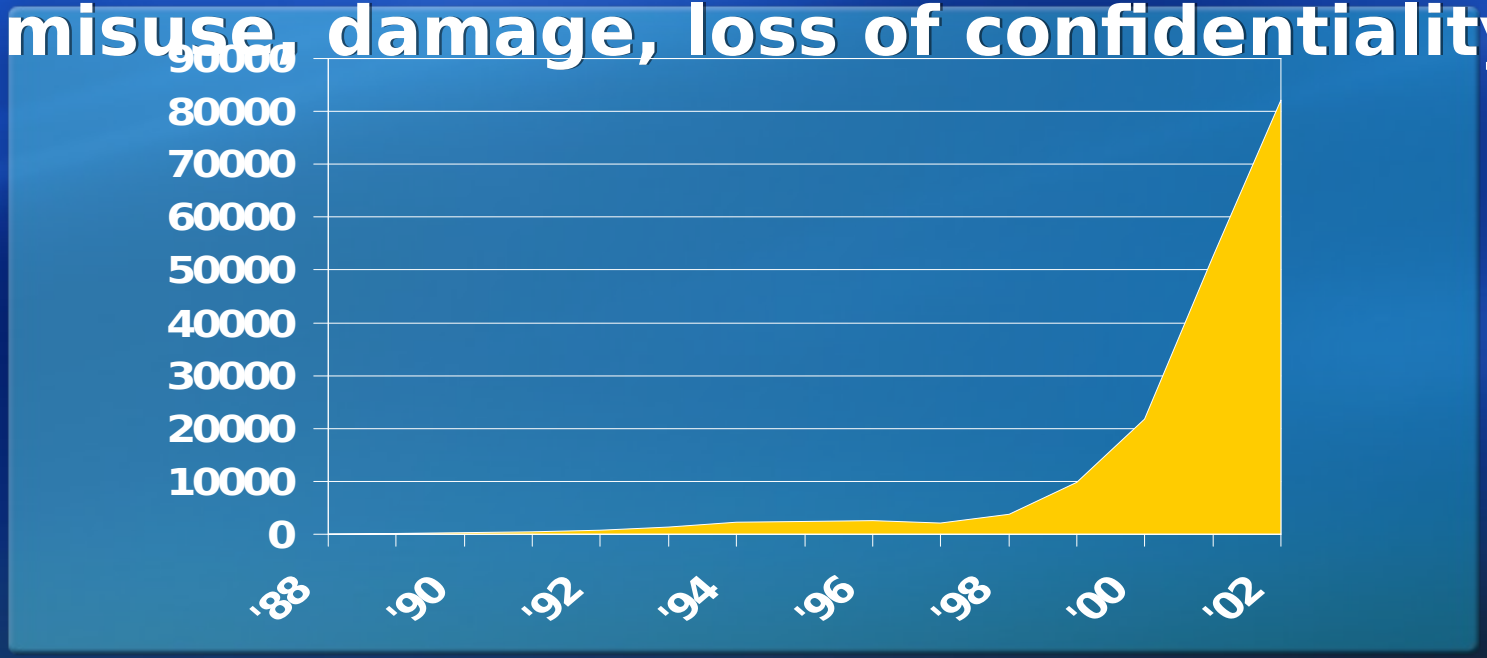
Microsoft

Agenda

- Risk Escalating Rapidly
  - SQL Injection Demo
- Case Study: SQL Server Security Push
  - SQL Server Lessons Learned
- Security Tools & Automation
- Admin, Data Protection, & App Design
- Summary

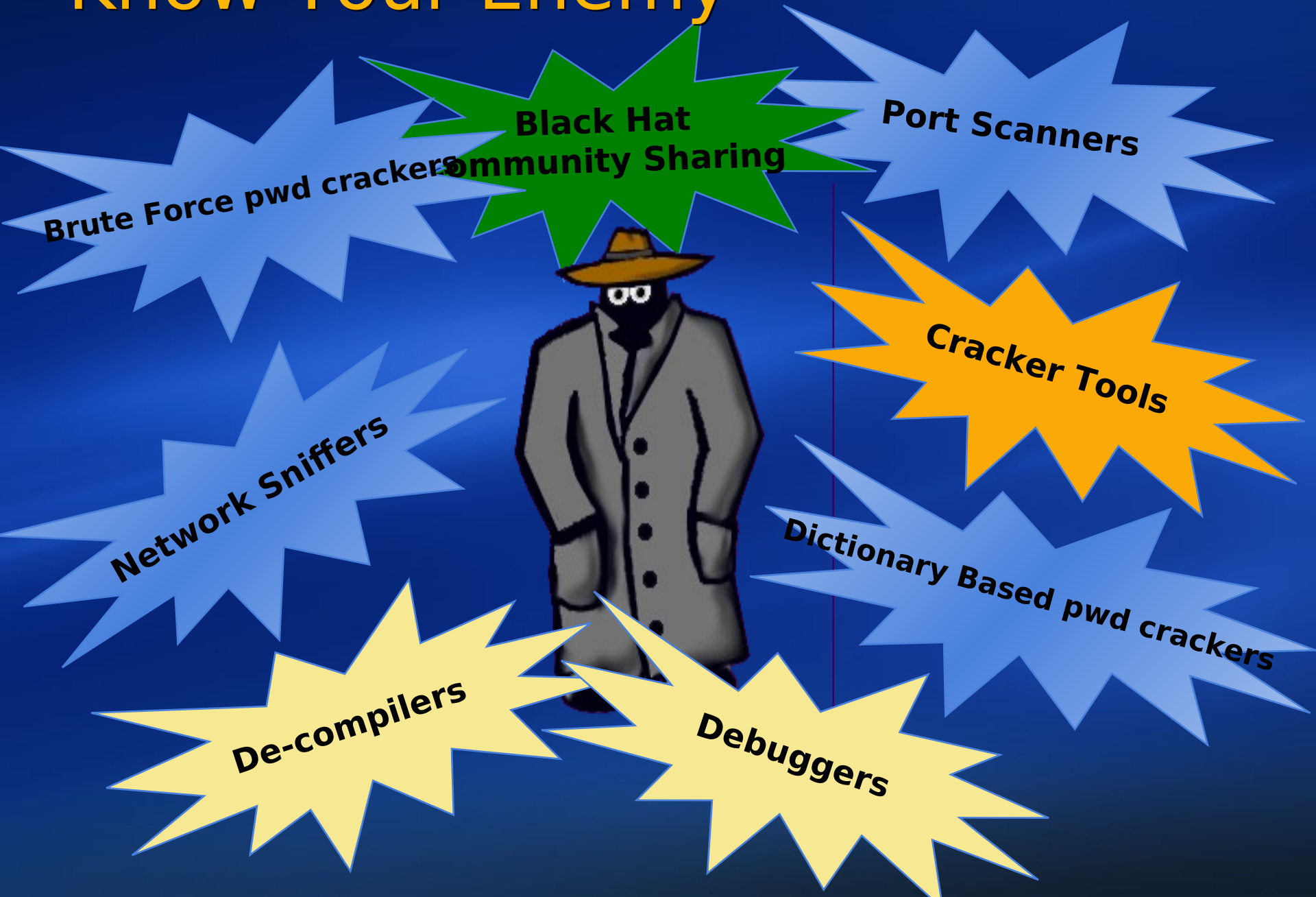
Incidents Reported (Industry wide)

- **CERT/CC incident statistics 1988 through 2003**
- ***Incident*: single security issue grouping together all impacts of that that issue**
- ***Issue*: disruption, DOS, loss of data, misuse, damage, loss of confidentiality**



Source: http://www.cert.org/stats/cert_stats.html

Know Your Enemy



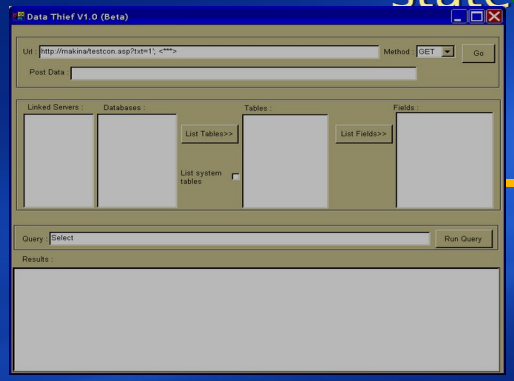
Data Thief Architecture

Attack string

Form values appended with extra SQL statement

SQL-Injected query

Contains an OPENROWSET statement



Vulnerable Application

App. Database

Local DB

SQL injected OPENROWSET statement causes remote DB to connect back to attackers DB, sending back useful

PDC03
Make the connection

Data Thief Demonstration

demo

Girish Chander
gchander@microsoft.com
SQL Server Security PM

Author: Cesar Cerrudo

Agenda

- Risk Escalating Rapidly
  - SQL Injection Demo
- Case Study: SQL Server Security Push
  - SQL Server Lessons Learned
- Security Tools & Automation
- Admin, Data Protection, & App Design
  - SQL Injection Demo
- Summary

Security Push Timeline

Push Preparation

- Goal full 800 person team productive from start
  - Identify Components
  - Complete threat models
  - Complete Education
- Select push start date
- Security plan
- Security reps from each team
- Set triage bars
- Infrastructure set-up

Security Push

- 5 million+ lines of code reviewed
- Two release in service
- One more release in dev
- 100% team focus during push
  - Dev, Test, PM, & UE
  - No other non-security work
- Three pronged approach:
  - Targeted code reviews
  - Tools targeting security
  - Threat driven reviews & testing

3/15/2003 5/1/2003 8/1/2003

Push Prep: Communications

- Learning from other teams' experiences
  - Windows, VS .Net, & IIS preceded SQL
- Team readiness critical
  - Don't start security push until team is prepared
- Security push plan
  - Motivation, goals, approach, process, fix bar,…
- Education plan for team
- Web site set up for general announcements and communication

Push Prep: Training

- Security training for every team member
  - Mandatory training for Architects, PMs, Developers and Testers
- Material covered includes:
  - Threat modeling, hacker/cracker tools, black hat community, security development & test tools, attack vectors & defense
- Video tape training for new team members
- Security talks series
  - more detail on important security related topics
  - Staying current with evolving threats
- On demand webcasts (search on security): <http://www.microsoft.com/usa/webcasts/ondemand/default.asp>

Push Prep: Infrastructure Ready

- Cross component team to drive push
  - SQL Security Leads
- Bug Tracking guidelines detailed
  - Classification of bugs and threats
- Separate bug tracking DB for tracking file reviews
  - Tracks code review progress & completeness
- Identification of components
  - 228 components; Risk level assessed for each
  - Threat models for each component
- Getting security tools running & building skills
- Clear fix criteria set
- Tracking progress is critical

Security Push Timeline

Push Preparation

- Goal full 800 person team productive from start
  - Identify Components
  - Complete threat models
  - Complete Education
- Select push start date
- Security plan
- Security reps from each team
- Set triage bars
- Infrastructure set-up

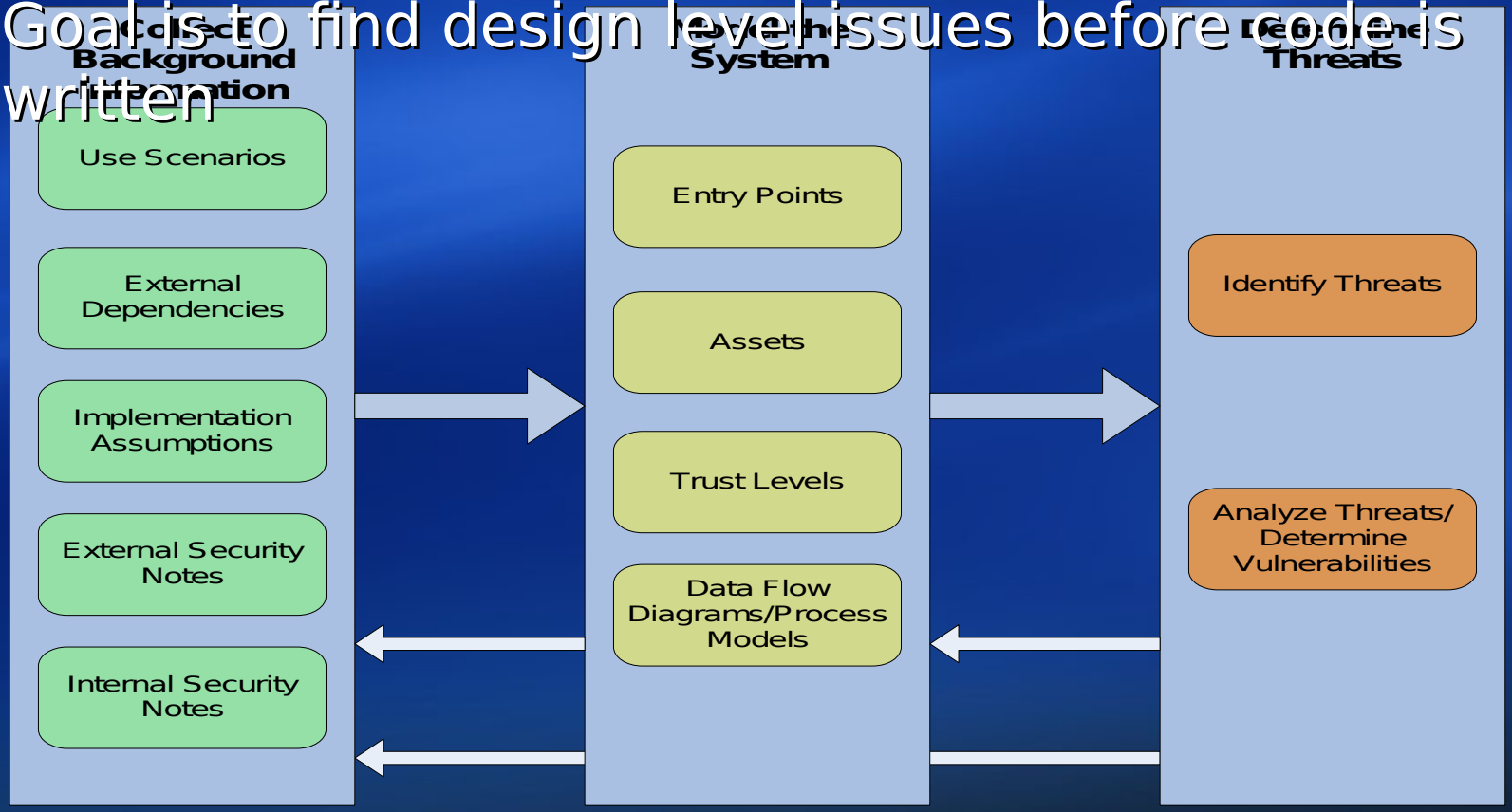
Security Push

- 5 million+ lines of code reviewed
- Two release in service
- One more release in dev
- 100% team focus during push
  - Dev, Test, PM, & UE
  - No other non-security work
- Three pronged approach:
  - Targeted code reviews
  - Tools targeting security
  - Threat driven reviews & testing

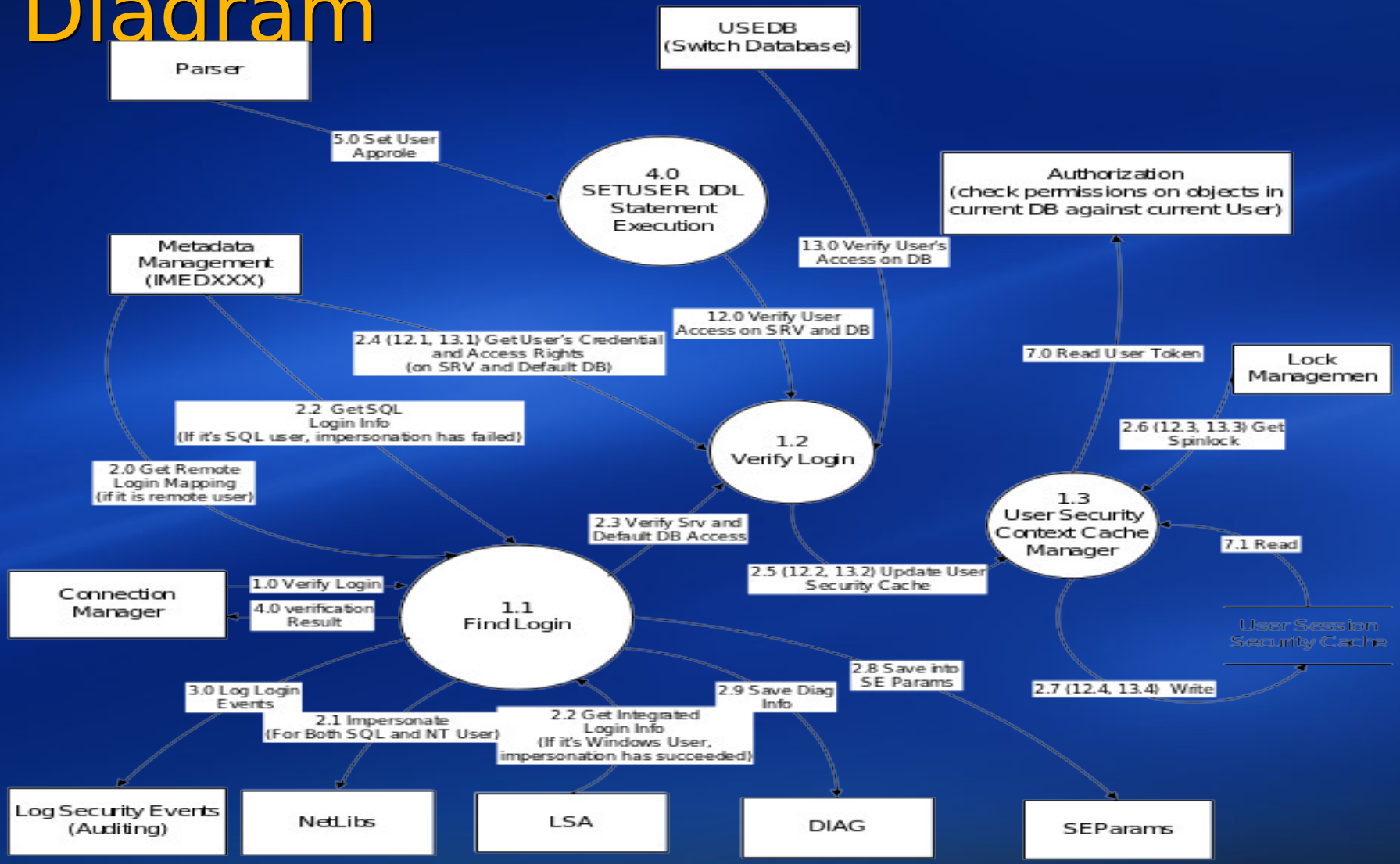
3/15/2003 5/1/2003 8/1/2003

Push: Threat Modeling Process

- A process to understand document threats to a system
  - Methodical and complete
  - Describes the system's threat profile
- Goal is to find design level issues before code is written



Push: Example Data Flow Diagram



Push: Threat Modeling

- Threats must be understood to build secure systems
- Every spec/design goes through threat analysis
  - Model of component is created (typically a DFD)
  - Threats categorized based on STRIDE
  - Severity ranked based on DREAD
    - NOT how hard it is to fix

S---Spoofing

T---Tampering of Data

R---Repudiation

I---information Disclosure

D---Denial of Service

E---Escalation of Privileges

D---Damage potential

R---Reproducibility

E---Exploitability

A---Affected Users

D---Discoverability

Push: Security SWAT Team

- Central team focused on cross component analysis
- Members chosen from different teams
- Build and share security expertise
- Overall Approach:
  - Met on daily basis
  - Choose component based on priority & risk
  - Invite relevant team members for that component
  - Collectively brainstorm to ferret out cross component threats
- Experience: an effective approach:
  - Part of ongoing, regular effort to audit product security

Push: Dead Code Removal

- Dead code removal
  - Code hygiene & work reduction
  - Why maintain & review non-executable code?
  - Code in product might be used in future
- Dead code detector built from code coverage tool
  - Analyzes compiled binaries
  - Automatically files bugs
    - One bug per file
    - Bug assigned to owner or last modifier

Push: Code Reviews

- Threat model directed & tools driven reviews
- Code review teams set up
  - Typically, 2 developers and 1 test at least
  - Code Review driver not code owner
  - Tester files bugs & scribe (some teams rotated roles)
- Code Review Experience:
  - Teams progressively became more efficient
  - First 90 minutes are the most effective
  - Pass of code by reviewer prior to code review helped
  - Presentation by code owner was very helpful

Push: Analytical Security Testing

A Testing method that simulates how an attacker operates

- Decompose the app (threat model driven)
- Identify interfaces
- Enumerate input points
  - Sockets
  - Pipes
  - Registry
  - Files
  - RPC (etc)
  - Command-line args
  - Etc.
- Enumerate data structures
  - C/C++ struct data
  - HTTP body
  - HTTP headers
  - HTTP header data
  - Other protocol headers
  - Querystrings
  - Bit flags
- Attack all data structures, wire formats, and input data

Push: Attack Team

- Red Team: Microsoft-wide ethical cracking group
- 50-50 split
  - Reactive: analysis of reported bugs
  - Proactive: security reviews
- Both formal and informal security reviews
  - Formal reviews by risk exposure
  - Greater exposure, deeper the review
- Analytical Security Testing
  - Advanced fuzz & data mutation

Security Push Timeline

Push Preparation

- Goal full 800 person team productive from start
  - Identify Components
  - Complete threat models
  - Complete Education
- Select push start date
- Security plan
- Security reps from each team
- Set triage bars
- Infrastructure set-up

Security Push

- 5 million+ lines of code reviewed
- Two release in service
- One more release in dev
- 100% team focus during push
  - Dev, Test, PM, & UE
  - No other non-security work
- Three pronged approach:
  - Targeted code reviews
  - Tools targeting security
  - Threat driven reviews & testing

Follow-on: What was learned?

- Set realistic schedules
- Get training done before starting
- Invest in tools early & aggressively
- Clearly identify system components early
- Code Reviews:
  - Provide guidelines & goals for each review
- Security focus improved overall system quality
  - Cross-component interactions better understood
  - Improved both functional & penetration testing
- Define an unambiguous exit criteria
- Clear progress tracking metrics required
- Process sometimes interferes with progress

Agenda

- Risk Escalating Rapidly
  - SQL Injection Demo
- Case Study: SQL Server Security Push
  - SQL Server Lessons Learned
- Security Tools & Automation
- Admin, Data Protection, & App Design
  - SQL Injection Demo
- Summary

Development Tools

- Engineers good at finding specific vulnerabilities
  - Innovation required
- Not good at reliably finding all instances of a specific bug class
  - Millions of lines of code
- Focus on tools to supplement manual efforts
  - Tools that can help identify issues in code
  - Managed code part of the answer
- Development tools used:
  - PREfix & PREfast
  - FXCop
  - Compiler options: /GS, SAFESEH
  - OS Level support: NOEXECUTE

PRE*fix* & PRE*fast*: Static Analysis

- Source code analysis:
  - Product independent coding issues
  - Product specific modules & filters
- Advantages:
  - Finds bugs even if not executed in test (unlike assert)
  - Can be run on each check-in or nightly
- Not magic bullets
- Many other techniques still required
  - Testing
  - Code reviews
  - Debuggers

Example Defect Classes

- Resource Leakage
  - Leaking Memory/Resource
- Pointer Management
  - Dereferencing NULL pointer
  - Dereferencing invalid pointer
  - Dereferencing or returning pointer to freed memory
- Illegal State
  - Resource in illegal state
  - Illegal value
  - Divide by zero
  - Writing to constant string
- Memory Management
  - Double frees
  - Freeing pointer to non-allocated memory (stack, global, etc.)
  - Freeing pointer in middle of memory block
- Initialization
  - Using uninitialized memory
  - Freeing or dereferencing uninitialized pointer
- Bounds violations
  - Overrun & Underrun
  - Failure to validate buffer size

• Managed code avoids many of these issues without post-authoring analysis tools

Agenda

- Risk Escalating Rapidly
  - SQL Injection Demo
- Case Study: SQL Server Security Push
  - SQL Server Lessons Learned
- Security Tools & Automation
- Admin, Data Protection, & App Design
- Summary

Application & DB Administration

- Basic security practices:
  - Automated enterprise software inventory
  - Run MBSA frequently
  - Apply latest patches
  - Use Windows Update or Software Update Service
- Audit authentication success & failures at all tiers
- Corporate security policy with periodic audit
  - Senior security Czar with ability to drive change
- Emergency response & disaster recovery plans
- Small admin group

Data Protection & App. Design

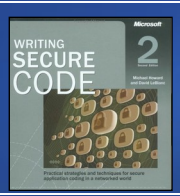
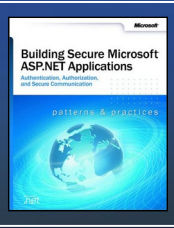
- Data Protection:
  - Hot standby: Clustering, log shipping, or DB Mirroring (Yukon)
  - Frequent backups: Offsite with media encryption
  - Offline, automated, non-production test systems
- Encrypted channels for transferring sensitive information
- Use integrated security with strong passwords
- Isolate Services
  - Do not install services on domain controller
  - Services should run under low privileged accounts (not shared)
  - Mid-tier/data-tier isolation with multiple firewalls
  - Surface area reduction: remove/disable unneeded services
- No direct access to data-tier
  - Two-tier client-side doesn't work – Security in data tier
    - Apps that "hide" DB passwords in client tier don't work
  - Access only via carefully reviewed mid-tier code

Summary

- Threat profile increasing
- SQL Security Push case study:
  - Communication, Training, Infrastructure & tools, Goals & exit criteria
- Security Tools and Techniques:
  - Threat models, Security SWAT team, Code reviews, Analytical security testing, Attack Team
- Application & DB Admin
- Data Protection & Application Design

Resources

- Microsoft Security and Privacy site
  - http://www.microsoft.com/security/
- SQL Security White paper
  - http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/maintain/security/sp3sec/Default.asp
- MBSA Home
  - http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/tools/mbsahome.asp

TITLE	
Microsoft Windows 2000 Security Technical Reference	
Writing Secure Code, 2/e	
Building Secure Microsoft® ASP.NET Applications	

PDC03

Make the connection

Solutions

Michael Howard, Microsoft
Matt Lyons, Microsoft
Stephanie Saad, Microsoft
Anthony Moore, Microsoft
Chris Wysopal, @Stake

Microsoft

Solutions and Tools

- Microsoft tools
  - Introducing a threat modeling tool
  - Code Access Security in a Nutshell
  - F5 in the Sandbox
  - FXCop
- Third-party tools
  - SafeApps

PDC03
Make the connection

Threat Modeling Tool

demo

Michael Howard
mikehow@microsoft.com
Senior Program Manager
Security Engineering &
Communications
Microsoft Corporation

PDC03

Make the connection

Code Access Security In A Nutshell

Matt Lyons
mattlyon@microsoft.com
Program Manager
Security Engineering & Communications
Microsoft Corporation

Microsoft

Running Shared Applications

- Many solutions today
  - ActiveX controls
  - Programs on file shares and web sites
  - Various scripting languages
  - And more...
- Struggle between safety and richness
  - ActiveX is rich, but hard to limit functionality
  - Scripting is generally safe, but not very rich

Managed Code Bridges The Gap

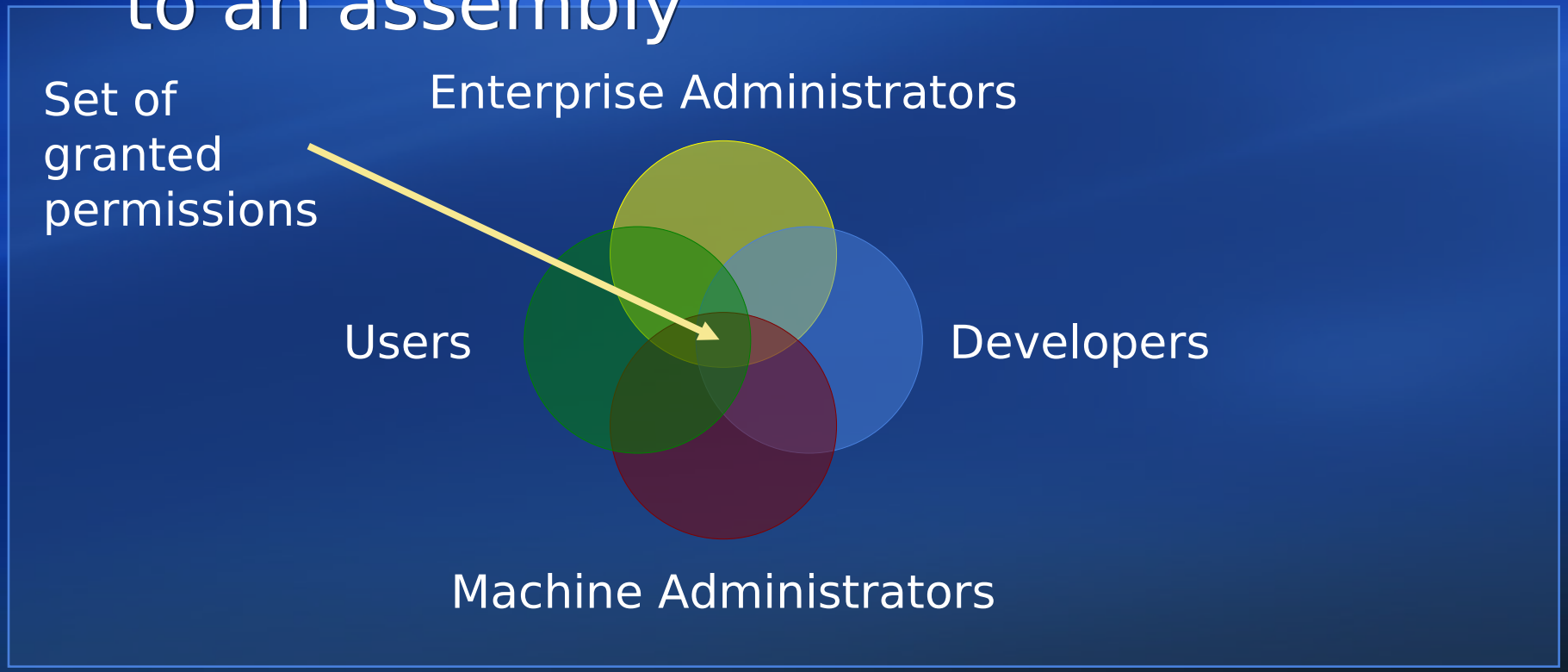
- Safety mechanisms
  - Code Access Security used by the .NET Framework to provide a sandbox
  - Fine-grained control of sandbox limits
- Richness
  - Built-in class libraries
  - Extensibility
  - Tool support

Fine-grained Permissions

- Permissions are “demanded” before accessing resources
- Resources protected by permissions
  - Files
  - Printing
  - Environment variables
  - Database access
  - Unmanaged code
  - And much more…

Distributed Policy Control

- Four contributors to policy
- Intersection of policy levels determines the permissions granted to an assembly



Default Policy Based On Zones

- MyComputer zone = all permissions
- Intranet zone = smaller set
- Internet zone = even smaller set

Click-Once Deployment

- New deployment technology
- Increases prevalence of application sandboxing
- Provides flexibility for granting permissions to sandboxed apps

PDC03
Make the connection

F5 in the Sandbox

demo

Stephanie Saad
ssaad@microsoft.com
Program Manager
Visual Studio
Microsoft Corporation

PDC03
Make the connection

FXCop

demo

Anthony Moore
amoore@microsoft.com
Development Lead
Common Language Runtime
Microsoft Corporation

SafeApps Automating the Security Code Review Process

PDC03
Make the connection

demo

Chris Wysopal
Director of Research & Development
@Stake

SafeApps™

- Secure Code Assurance (SCA) engine
  - Replaces a manual security code review. @stake expert code reviewer in a box.
- Detects the programming errors that lead to security vulnerabilities. Assists in remediating the errors.
  - Detects programming errors that lead to viruses and worms
  - Prioritizes risk of each error from *severe error to warning.* Optimizes programmer's time.
  - Guides the programmer to fix the source of error. Most programmers don't know how to fix security errors.
- Target user
  - Developer, QA Engineer, Security Engineer
- Development teams that use SafeApps can drastically reduce the number of vulnerabilities in their software

What is unique about SafeApps?

- @stake's world class application experience in a box
  - Expert code reviewers on our development team
- Extensible scripted architecture
  - Can update with new script packages that detect newly found classes of problems
  - Can build script packs tailored to particular customer environments
- Detects vulnerabilities as early as possible for maximum security ROI.
- Analysis performed on program binaries instead of the source code
  - Deepest security analysis possible
  - Uses the context of the entire program
  - Evaluates interaction with OS and other binary components
- Risk Analysis Reporting
  - Summarizes overall program risk. Can be rolled up for an entire enterprise
  - Prioritizes errors by risk. Programmers can fix highest risk problems first.

Detecting security coding flaws

- SafeApps modeling engine builds control flow and data flow graphs of the program. Range of data is propagated.
- Scripts analyze the graphs for coding flaws
  - Language and standard library issues
    - Buffer overruns (off by ones, size mismatches), format string vulnerabilities, integer overflows (type conversions), race conditions, error return checking
  - Platform API
    - Privilege escalation, cryptography usage, database usage, network usage
  - High level issues
    - Backdoors, denial of service, HTTP, input validation

Summary

- There is no silver bullet for producing secure code
- Threats are evolving and your code must withstand new attacks
- Education is key

Call to Action

- Create your own Security Awareness Program
- Do a Security Push
- Utilize tools & best practices
- Test on Windows XP SP2 & Longhorn
- Attend the Security Symposium panel
  - Carl Ellison, Howard Schmidt, Chris Wysopal, Jason Garms, James Hamilton
  - First 800 to return for the panel will receive “Improving Web Applications Security”
- Complete the security survey
  First 800 to turn it in will receive a cool new “spy pen”

PDC03

Make the connection

Microsoft®